



Dedukti:un vérificateur de preuves universel

Ali Assaf, Raphaël Cauderlier, Ronan Saillard

► To cite this version:

Ali Assaf, Raphaël Cauderlier, Ronan Saillard. Dedukti:un vérificateur de preuves universel. Journées nationales GDR - GPL - CIEL - AFADL, Apr 2013, Nancy, France. hal-01086609

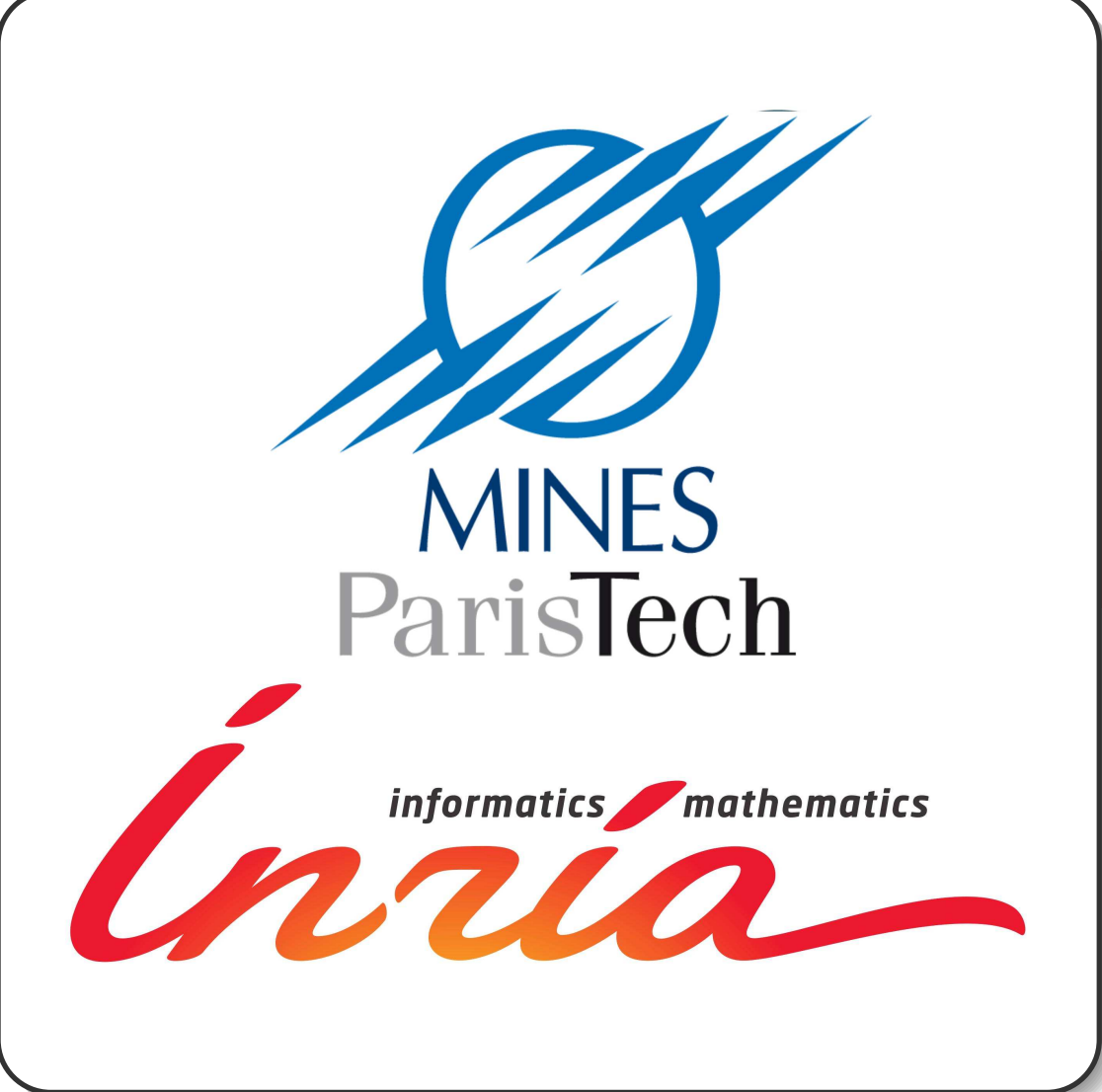
HAL Id: hal-01086609

<https://hal-mines-paristech.archives-ouvertes.fr/hal-01086609>

Submitted on 24 Nov 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Dedukti: un vérificateur de preuves universel

Ali Assaf, Raphaël Cauderlier et Ronan Saillard

DEDUCTEAM (INRIA) - MINES ParisTech

ali.assaf@inria.fr raphael.cauderlier@inria.fr ronan.saillard@inria.fr

Introduction

DEDUKTI est un vérificateur de types pour le $\lambda\Pi$ -calcul modulo, un formalisme alliant types dépendants et réécriture qui permet d'exprimer et de vérifier les preuves de nombreux systèmes logiques.

Nous proposons d'utiliser DEDUKTI comme un vérificateur de preuves universel en traduisant *HOL*, *Coq* et *FoCaLize* vers DEDUKTI.

HOL

```
# let transitivity =
  EQ_MP
  (MK_COMB (REFL '(=) x : A -> bool',
    ASSUME 'y : A = z'))
  (ASSUME 'x : A = y');;

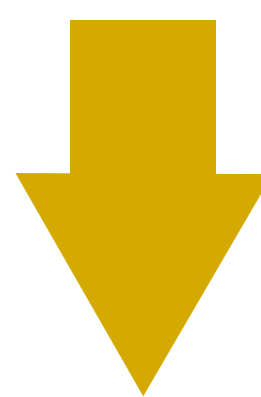
val transitivity :
  thm = x = y, y = z |- x = z
```

Coq

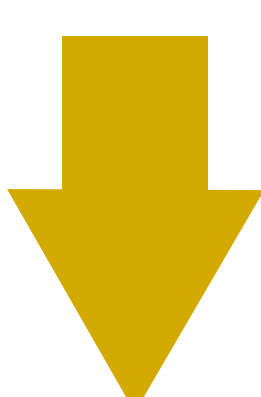
```
Theorem transitivity :
  forall (A : Type) (x y z : A),
    x = y -> y = z -> x = z.
Proof.
  intros A x y z H1 H2.
  induction H1.
  exact H2.
Qed.
```

FoCaLize

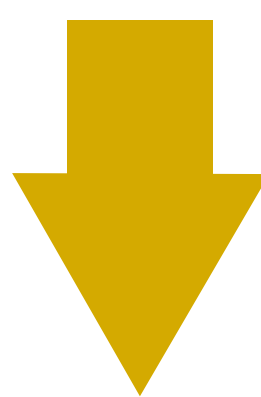
```
species Setoid =
  signature (=): Self -> Self -> bool;
  property transitivity :
    all x y z : Self,
      x = y -> y = z -> x = z;
end;;
```



Holide



Coqine



Focalide

Dedukti

```
A : Type.
Nat: Type.
Z : Nat.
S : Nat -> Nat.

plus: Nat -> Nat -> Nat.
[m:Nat] plus Z m --> m
[n:Nat,m:Nat] plus (S n) m --> plus n (S m).

Listn : Nat -> Type.
nil : Listn Z.
cons : n:Nat -> A -> Listn n -> Listn (S n).

append: n:Nat -> Listn n -> m:Nat -> Listn m -> Listn (plus m n).
[n:Nat,l1:Listn n] append n l1 Z nil --> l1
[n:Nat,l1:Listn n,m:Nat,l2:Listn m,a:A]
  append n l1 (S m) (cons {m} a l2) --> append (S n) (cons n a l1) m l2.
Définition de la concaténation de listes de taille n en Dedukti.
```

- Un **vérificateur de preuves** basé sur le $\lambda\Pi$ -calcul modulo (voir encadré).
- Une architecture originale : le fichier source est **compilé** vers un langage cible (*Lua*) qui est ensuite **exécuté** pour obtenir le résultat.
- De la **normalisation par évaluation** : la normalisation est assurée par l'exécution dans le langage cible.
- Une compilation **just-in-time (JIT)** : l'utilisation d'un compilateur *JIT* comme *back-end* assure des performances optimales quelque soit la quantité de calcul dans les preuves vérifiées.
- Une vérification de type **sans contexte** : le contexte est remplacé par des annotations de type.
- Un algorithme **bi-directionnel** : le système alterne entre des phases de vérification et d'inférence de type, guidé par la structure du terme.

|

>

Dedukti

génération du code

>

Lua

exécution du code

OK

>

KO

Runtime

dedukti.lua

Dedukti est un générateur de code.

Le $\lambda\Pi$ -calcul modulo

- Le $\lambda\Pi$ -calcul est un λ -calcul avec des **types dépendants** qui permet d'exprimer les preuves de la logique minimale des prédicats à travers la correspondance de Curry-DeBruijn-Howard.
- Le $\lambda\Pi$ -calcul modulo est une extension du $\lambda\Pi$ -calcul qui intègre une notion de convertibilité élargie :

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash B : s \quad A \equiv_{\beta\mathcal{R}} B}{\Gamma \vdash t : B} (\text{Conv})$$

où \mathcal{R} est la congruence générée par un système de **réécriture** bien typé arbitraire.

Vers l'interopérabilité

- Les systèmes de preuves actuels souffrent d'un manque d'**interopérabilité**. Il est difficile de réutiliser une théorie d'un système dans un autre sans refaire toutes les preuves.
- La traduction de ces différents systèmes dans un formalisme commun permettra de **combiner** leurs preuves pour construire des théories plus larges.
- Des traductions d'autres systèmes, tels que *PVS*, *Matita* ou encore *Atelier B*, sont à l'étude.